GeneSelectR Application on TCGA-BRCA Dataset

Damir Zhakparov

2023-10-11

Contents

Introduction **1**

Biological Question . . . 2

Dataset Overview . . . 2

Molecular Subtypes Overview . . . 2

1. Differential Gene Expression Analysis **2**

2. Feature Selection with GeneSelectR **4**

2.1 Data Preparation . . . 4

2.2 Running GeneSelectR . . . 4

3. Results Inspection **5**

3.1 Machine Learning Performance Metrics . . . 5

3.2 Feature Importance . . . 6

3.3 Overlap between Gene Lists and DGE list . . . 14

3.4 Gene Lists Annotation . . . 14

3.5 Gene Ontology Enrichment . . . 15

3.6 Quantification of Children Nodes of Parent Node of Interest . . . 20

3.7 Semantic Similarity Analysis . . . 24

4. Picking the Winner and Conclusion **26**

Introduction

This tutorial walks you through the use of `GeneSelectR` with the **TCGA-BRCA RNA Expression dataset** from The Cancer Genome Atlas. To acquire the dataset, we employed the `TCGAbiolinks` R package. For a step-by-step guide on data extraction, consult this script.

Biological Question

The primary aim of this tutorial is to identify a transcriptomic signature specific to each molecular subtype of breast cancer as defined by PAM50 markers. By doing so, we hope to shed light on the unique molecular mechanisms underlying each subtype, which could subsequently inform targeted therapeutic strategies and prognostic assessments.

Dataset Overview

In this guide, we focus on a subset of **380 samples**, all categorized under the **'Primary Solid Tumor'** type and labeled with **PAM50 markers**: Basal(n = 80), Her2(n = 38), LumA(n = 188), and LumB (n = 74). The analysis encompasses all **60,600 sequenced transcripts**. For an in-depth look at the TCGA-BRCA dataset, please visit the official documentation. The data files used in the tutorial can be accessed here.

Molecular Subtypes Overview

Basal-like

- **Characteristics**: Triple Negative (HR-, HER2-), high levels of Ki-67
- **Prognosis**: Poor
- **Common Treatments**: Chemotherapy

HER2-enriched

- **Characteristics**: HER2 Positive (HER2+), Hormone Receptor Negative (HR-)
- **Prognosis**: Intermediate
- **Common Treatments**: HER2-targeted therapies like trastuzumab

Luminal A

- **Characteristics**: Hormone Receptor Positive (HR+), low levels of HER2 and Ki-67
- **Prognosis**: Best among the subtypes
- **Common Treatments**: Hormone therapy

Luminal B

- **Characteristics**: Hormone Receptor Positive (HR+), higher levels of HER2 and Ki-67
- **Prognosis**: Worse than Luminal A but better than Basal-like
- **Common Treatments**: Hormone therapy, may require chemotherapy

By understanding the unique transcriptomic landscape of each subtype, we can better predict disease outcomes and tailor treatment regimens.

1. Differential Gene Expression Analysis

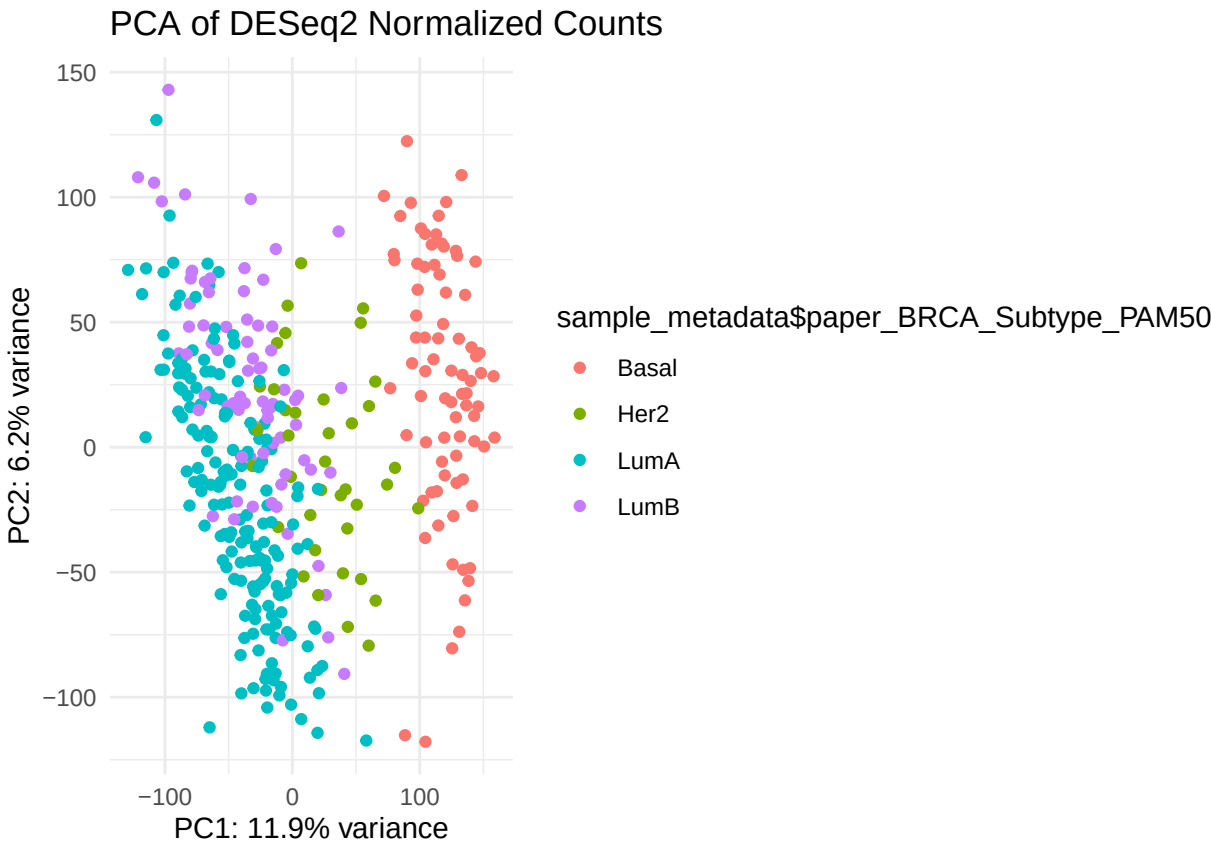
As a baseline differential gene expression analysis (DGE) provides a good starting point. First of all we will load the dataset and metadata files and import necessart packages:

```r
# set up working directories
data_dir <- file.path('../raw-data')
output_dir <- file.path('../results')

#load the files
sample_metadata <- readRDS(file.path(data_dir, 'sample_metadata.rds'))
raw_counts <- readRDS(file.path(data_dir, 'raw_counts.rds'))
```

After that we will create a DESEq2 object and then filter out genes with low counts:

```r
dds <- DESeqDataSetFromMatrix(countData = raw_counts,
                              colData = sample_metadata,
                              design = ~ paper_BRCA_Subtype_PAM50)

# filter the low counts
table(sample_metadata$paper_BRCA_Subtype_PAM50)
#>
#> Basal  Her2  LumA  LumB
#>    80    38   188    74
smallestGroupSize <- 38 #smallest group is 38 samples
keep <- rowSums(counts(dds) >= 10) >= smallestGroupSize
dds <- dds[keep,]
```

Before performing the DGE analysis it's strongly advised to perform PCA on normalized counts to see if there any batch effects. We can do it by running:



And then plotting:

And now let's run the DGE analysis:

```
# Run DESeq
dds <- DESeq(dds)
```

After the calculations are done, let's filter out differentially expressed genes:

```
# Get DE results
res <- results(dds)
filtered_res <- subset(res, padj < 0.001 & abs(log2FoldChange) > log2(5))
filtered_df <- as.data.frame(filtered_res)
```

2. Feature Selection with GeneSelectR

2.1 Data Preparation

GeneSelectR expects the data to be a matrix/dataframe where rows represent samples and columns are features. Please note that the dataset has to be normalized with between sample normalization prior to any analysis with the package. So now we extract the vst-transformed matrix from our DESeq2 object for the feature selection:

```
# Extract VST (Variance-Stabilized Transformed) Data
vsd <- vst(dds, blind = FALSE)
vsd_matrix <- assay(vsd)
vsd_matrix <- t(vsd_matrix)
```

Then we will create a response vector y with sample labels:

```
sample_metadata <- sample_metadata[match(rownames(exp),rownames(sample_metadata)), ]
sample_metadata$paper_BRCA_Subtype_PAM50 <- as.factor(sample_metadata$paper_BRCA_Subtype_PAM50)
sample_metadata$num_label <- as.integer(sample_metadata$paper_BRCA_Subtype_PAM50) # NOTE: the labels sh
table(sample_metadata$num_label)
#> < table of extent 0 >
```

2.2 Running GeneSelectR

After preparing the data we are ready to run GeneSelectR:

```
X <-  exp
y <- sample_metadata %>% select(num_label)

selection_results <- GeneSelectR(X, # gene expression matrix
                                 y, # label vector
                                 njobs = -1, # number of cores to de deployes (-1 = all)
                                 n_splits = 5, # number of train/test splits to perform
                                 max_features = 250, # max futures to be selected per FS method (only R
                                 perform_test_split = TRUE, # if partition data into train and test
                                 scoring = 'accuracy', # scoring metric for optimization
                                 calculate_permutation_importance = TRUE, # whether to calculate permut
                                 search_type = 'random', # type of grid search
                                 n_iter = 150L) # number of hyperparameter combinations to be samples
```

3. Results Inspection

After the analysis has been finished, we can look into the PipelineResults objects to inspect the results. For example if we call:

```
str(selection_results, max.level = 1)
#> Formal class 'PipelineResults' [package "GeneSelectR"] with 6 slots
```

We can see following slots in the Pipeline results object: - best_pipeline - contains all the parameters for the best performing pipeline; - cv_results - contains the entire output of the CV procedure during hyperparameter search; - inbuilt_feature_importance - inbuilt feature importance with mean, std and rank for every feature across iterations;
- permutation_importance - permutation feature importance with mean, std and rank for every feature across iterations;
- cv_mean_scores - CV scores but agreggated into one dataframe with mean and sd of the optimization metrics scores;
- test_metrics - metrics (f1, recall, precision and accuracy) for the unseen test set. Let's inspect some of the most interesting metrics for the evaluation.

3.1 Machine Learning Performance Metrics

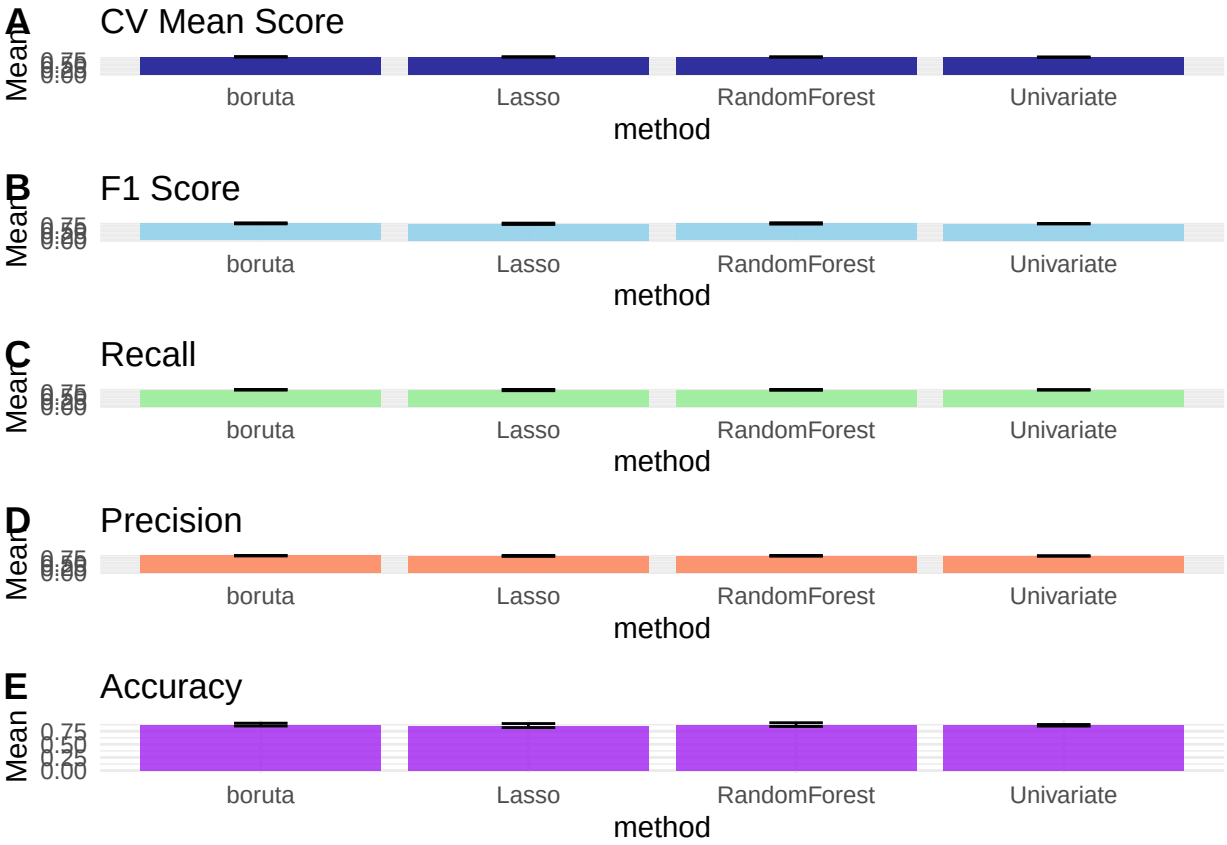
First, let's explore the cross validation scores for every of our methods. We can inspect the CV mean performance by displaying the dataframe:

```
selection_results@cv_mean_score
#>          method mean_score    sd_score
#> 1       boruta  0.9210374 0.012047342
#> 2        Lasso  0.9153061 0.007984372
#> 3 RandomForest  0.9127721 0.007926976
#> 4   Univariate  0.9045578 0.011999833
```

We can see that boruta is slightly better in terms of CV performance, but all methods are somewhat comparable. In this case we can inspect the performance on unseen data that is stored in test_metrics slot:

```
selection_results@test_metrics
#> # A tibble: 4 x 9
#>   method       f1_mean  f1_sd recall_mean recall_sd precision_mean precision_sd
#>   <chr>          <dbl>  <dbl>       <dbl>     <dbl>          <dbl>        <dbl>
#> 1 Lasso          0.854 0.0351       0.858    0.0388          0.870       0.0262
#> 2 RandomForest   0.873 0.0346       0.876    0.0356          0.883       0.0286
#> 3 Univariate     0.859 0.0127       0.863    0.0150          0.866       0.0136
#> 4 boruta         0.876 0.0251       0.876    0.0273          0.889       0.0234
#> # i 2 more variables: accuracy_mean <dbl>, accuracy_sd <dbl>
```

Again, boruta seems to be the best one, although marginally. We can produce a combined plot of all metrics

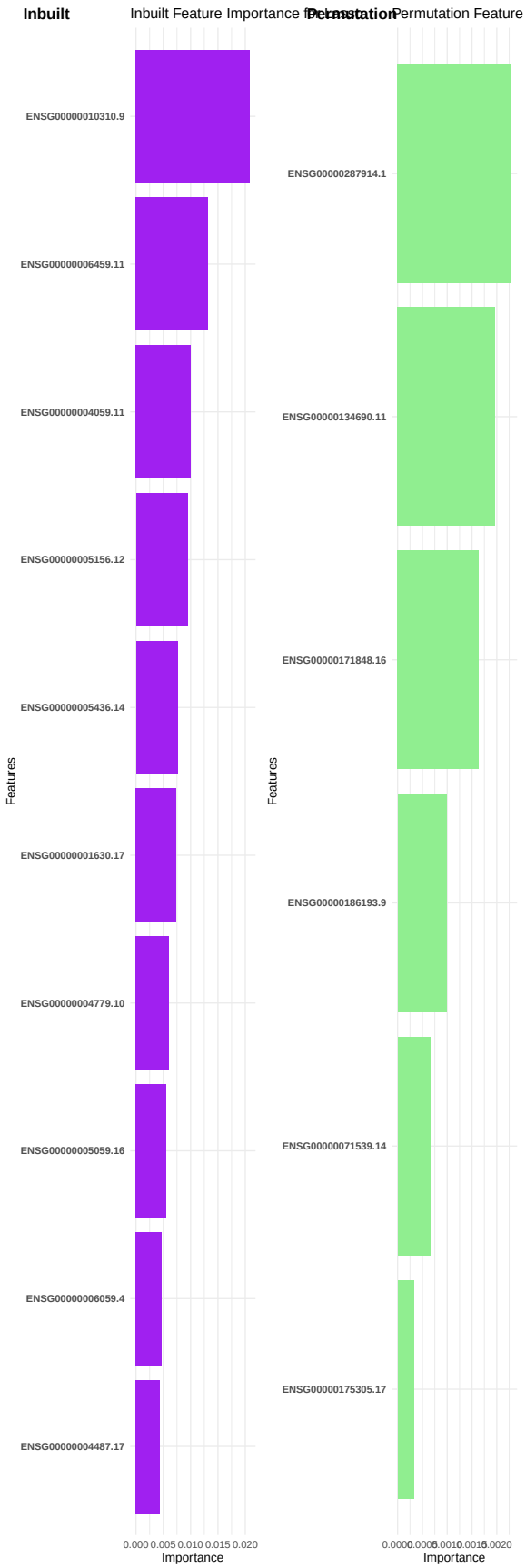


by calling:

3.2 Feature Importance

The next step is to inspect what are the most important features for every feature selection method. To plot the importance scores we will call the plot_feature_importance(). The function returns a list of plots demonstrating mean feature importance scores across different data splits, so we will store it in a separate object:

```
plot_list <- plot_feature_importance(selection_results)
plot_list
#> $Lasso
```


Inbuilt
Inbuilt Feature Importance for Leukemia
Permutation
Permutation Feature
ENSG00000010310.9
ENSG00000006459.11
ENSG00000004059.11
ENSG00000005156.12
ENSG00000005436.14
ENSG00000001630.17
ENSG00000004779.10
ENSG00000005059.16
ENSG00000006059.4
ENSG00000004487.17
ENSG00000287914.1
ENSG00000134690.11
ENSG00000171848.16
ENSG00000186193.9
ENSG00000071539.14
ENSG00000175305.17
Features
Features
0.000 0.005 0.010 0.015 0.020
0.0000 0.0005 0.0010 0.0015 0.0020
Importance
Importance

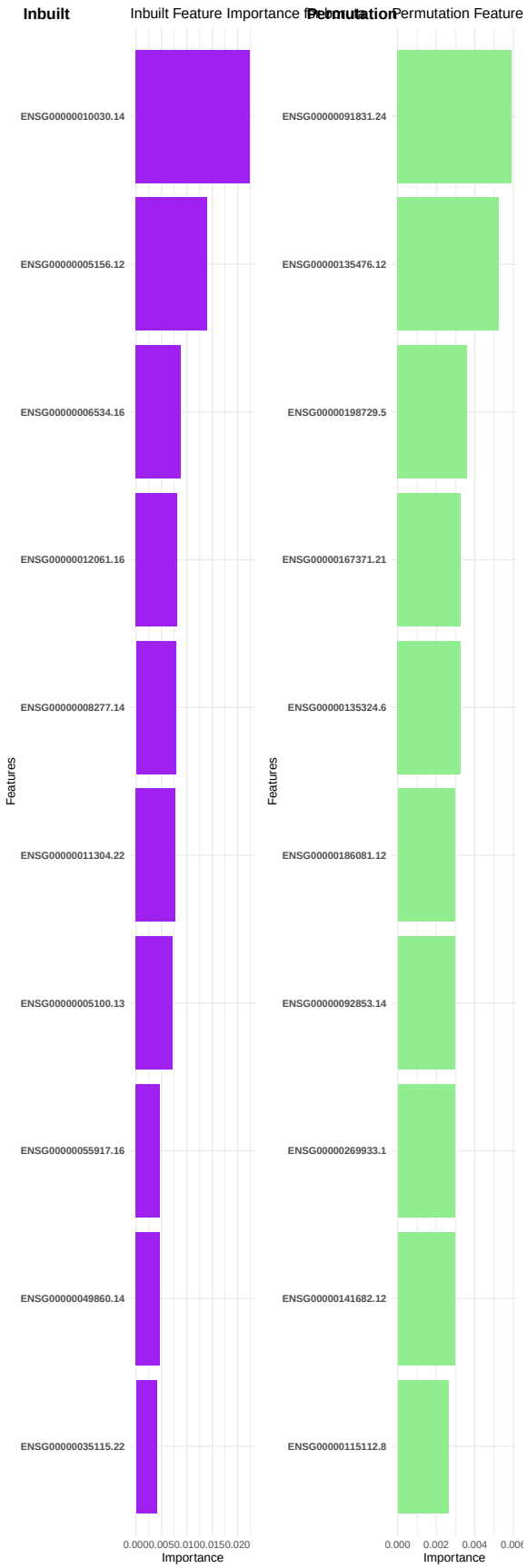
```
#>
#> $Univariate
```


Inbuilt
Inbuilt Feature Importance
Permutation
Permutation Feature
ENSG00000002586.20
ENSG00000002549.12
ENSG00000004399.13
ENSG00000004142.12
ENSG00000006016.11
ENSG00000008311.15
ENSG00000007516.14
ENSG00000002726.21
ENSG00000007384.15
ENSG00000007264.15
Features
0.000 0.002 0.004 0.006
Importance
ENSG00000054598.9
ENSG00000091651.9
ENSG00000123485.12
ENSG00000143228.13
ENSG00000182175.14
ENSG00000075618.18
ENSG00000148468.17
ENSG00000089101.19
ENSG00000254290.1
ENSG00000204531.20
Features
0.000 0.002 0.004 0.006 0.008
Importance

```
#>
#> $RandomForest
```


Inbuilt
Inbuilt Feature Importance for Random Forest
Permutation
Permutation Feature
ENSG00000002745.13
ENSG00000006025.12
ENSG00000008988.11
ENSG00000003393.15
ENSG00000005001.10
ENSG00000007923.16
ENSG00000005108.16
ENSG00000008710.20
ENSG00000010295.20
ENSG00000004478.8
ENSG00000143228.13
ENSG00000066279.18
ENSG00000287914.1
ENSG00000138180.16
ENSG00000148773.14
ENSG00000186185.14
ENSG00000117724.13
ENSG00000007968.7
ENSG00000183856.11
ENSG00000106462.11
Features
Features
0.000
0.005
0.010
0.000
0.002
0.004
0.006
Importance
Importance

```
#>
#> $boruta
```


Inbuilt
Inbuilt Feature Importance
Permutation
Permutation Feature
ENSG00000010030.14
ENSG00000005156.12
ENSG00000006534.16
ENSG00000012061.16
ENSG00000008277.14
ENSG00000011304.22
ENSG00000005100.13
ENSG00000055917.16
ENSG00000049860.14
ENSG00000035115.22
ENSG00000091831.24
ENSG00000135476.12
ENSG00000198729.5
ENSG00000167371.21
ENSG00000135324.6
ENSG00000186081.12
ENSG00000092853.14
ENSG00000269933.1
ENSG00000141682.12
ENSG00000115112.8
Features
Features
0.000 0.005 0.010 0.015 0.020
0.000 0.002 0.004 0.006
Importance
Importance

Interestingly, we can see that boruta has a lot of relevant genes to cancer as top features: ENSG00000010030 (ETV7),
ENSG00000005156 (LIG3), ENSG00000006534 (ALDH3B1), ENSG00000005100 (DHX33).

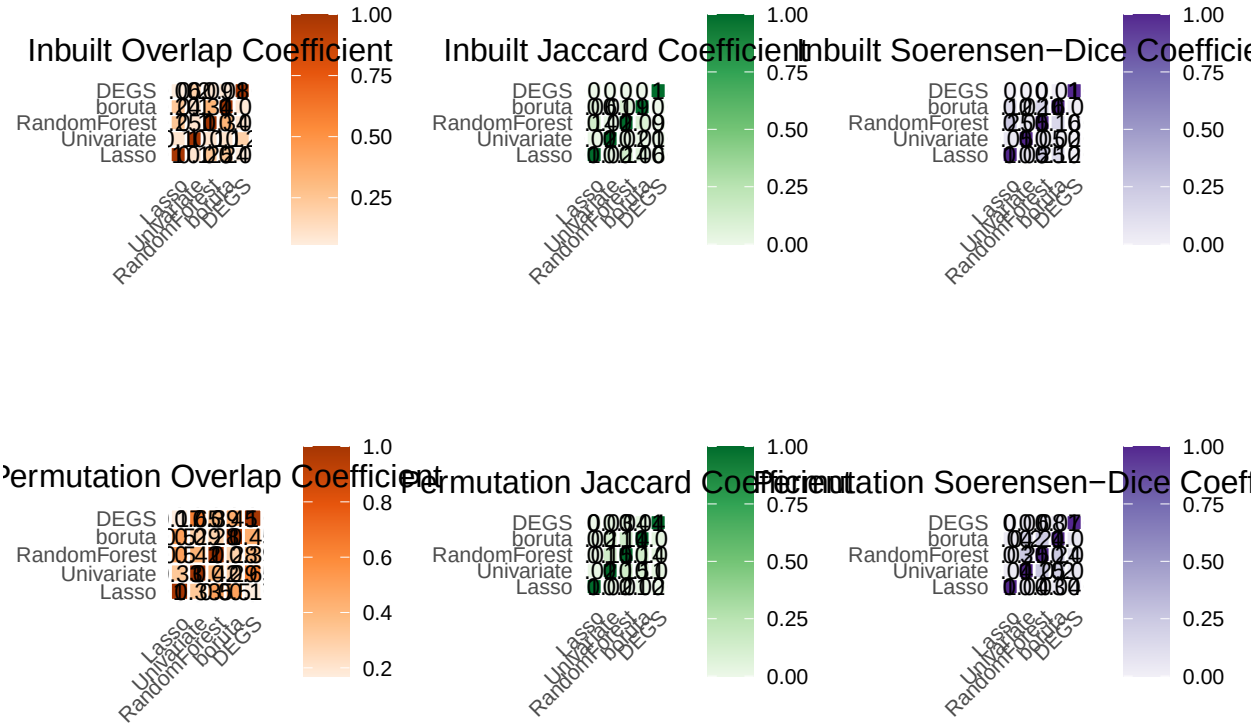
3.3 Overlap between Gene Lists and DGE list

It might be interesting to see if there is any overlap between the feature selection lists also including the list of differentially expressed genes. Let's first extract the list of DEGs:

```
deg_list <- rownames(filtered_df)
```

Then we can calculate the overlap coefficients and plot them:

```
#inspect overlap with DEGs
overlap_degs <- calculate_overlap_coefficients(selection_results, custom_lists = list('DEGS' = deg_list)
plot_overlap_heatmaps(overlap_degs)
```



We can see three different coefficients that calculate the list a bit differently. Since our lists are different in sizes, the most relevant one here is Overlap Coefficient. We can see that some of the permutation importance lists have similarities. For example, Lasso is quite similar to boruta and RandomForest. As for the DEGs list it's quite similar to the Univariate method, although this is somewhat expected.

3.4 Gene Lists Annotation

For the enrichment analysis and subsequent biological interpretation it is convenient to convert gene identifiers to others that are appropriate/useful in different situations. To do so we can do the following:

```
# remove version number from the ensembl ids
background <- as.character(colnames(vsd_matrix))
background <- gsub("\\..*", "", background)
custom_list <- list('background' = background,
                    'DEGs' = deg_list)

ah <- AnnotationHub::AnnotationHub()
human_ens <- AnnotationHub::query(ah, c("Homo sapiens", "EnsDb"))
human_ens <- human_ens[['AH98047']]
#> loading from cache
#> require("ensembldb")
annotations_ahb <- ensembldb::genes(human_ens, return.type = "data.frame") %>%
  dplyr::select(gene_id,gene_name,entrezid,gene_biotype)

annotations_df <- annotate_gene_lists(pipeline_results = selection_results,
                                      annotations_ahb = annotations_ahb,
                                      format = 'ensembl_id',
                                      custom_lists = custom_list)
#> 'select()' returned 1:1 mapping between keys and columns
#> 'select()' returned 1:1 mapping between keys and columns
#> 'select()' returned 1:1 mapping between keys and columns
#> 'select()' returned 1:1 mapping between keys and columns
#> 'select()' returned 1:many mapping between keys and columns
#> 'select()' returned 1:many mapping between keys and columns
#> 'select()' returned 1:1 mapping between keys and columns
#> 'select()' returned 1:1 mapping between keys and columns
#> 'select()' returned 1:1 mapping between keys and columns
#> 'select()' returned 1:many mapping between keys and columns
#> 'select()' returned 1:many mapping between keys and columns
#> 'select()' returned 1:many mapping between keys and columns
```

This returns an object of class AnnotatedGeneLists containing gene symbol, ENSEMBL ID and ENTREZ ID for the selected features as well as background list and DEGs list.

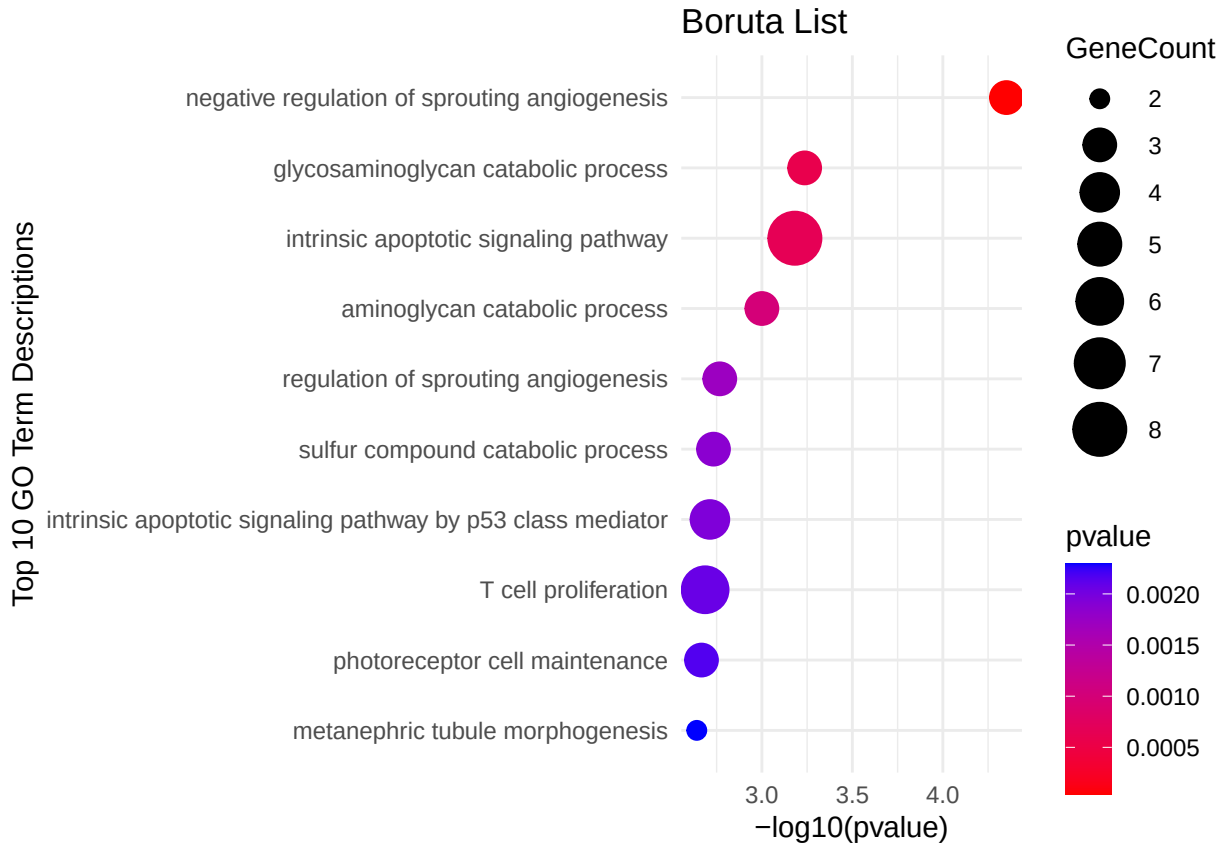
3.5 Gene Ontology Enrichment

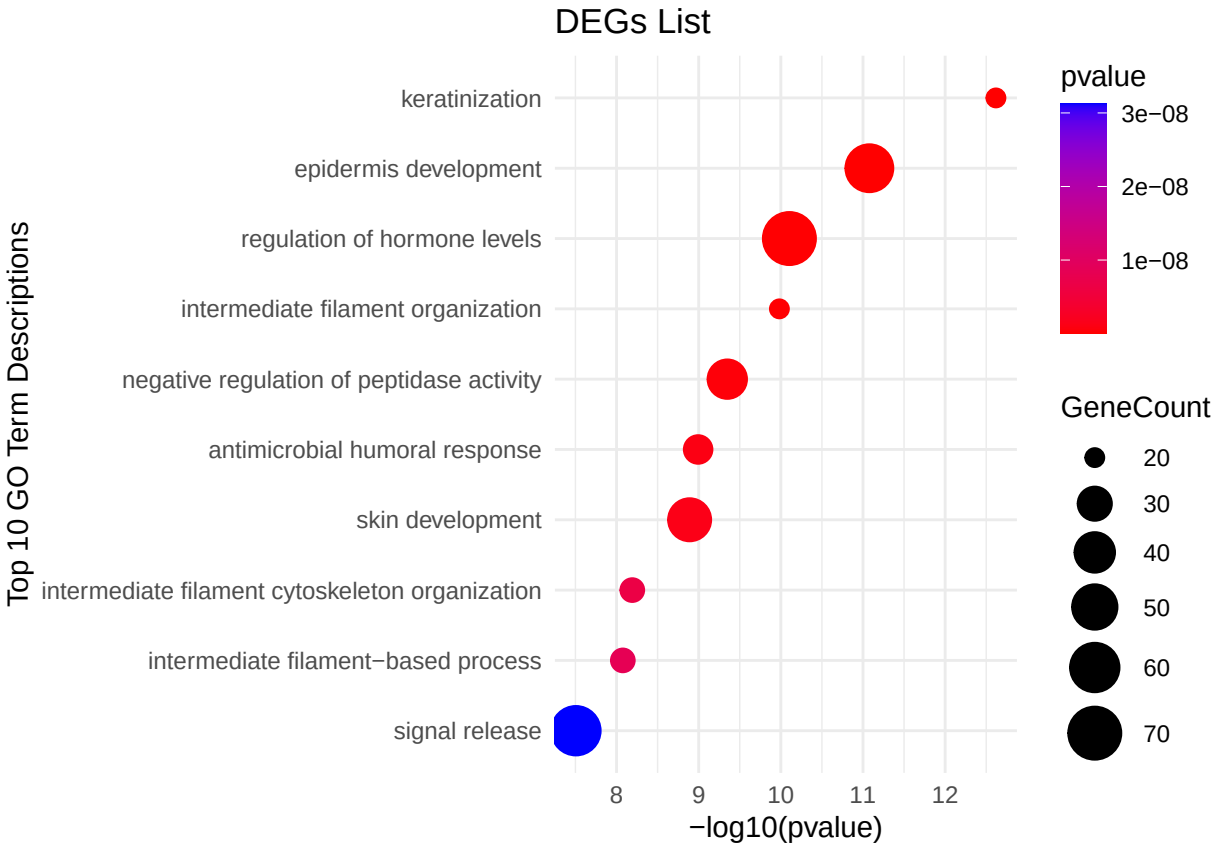
Now we will do Gene Ontology Enrichment analysis to mine the pathways relevant to our biological question. To do so we can call the function:

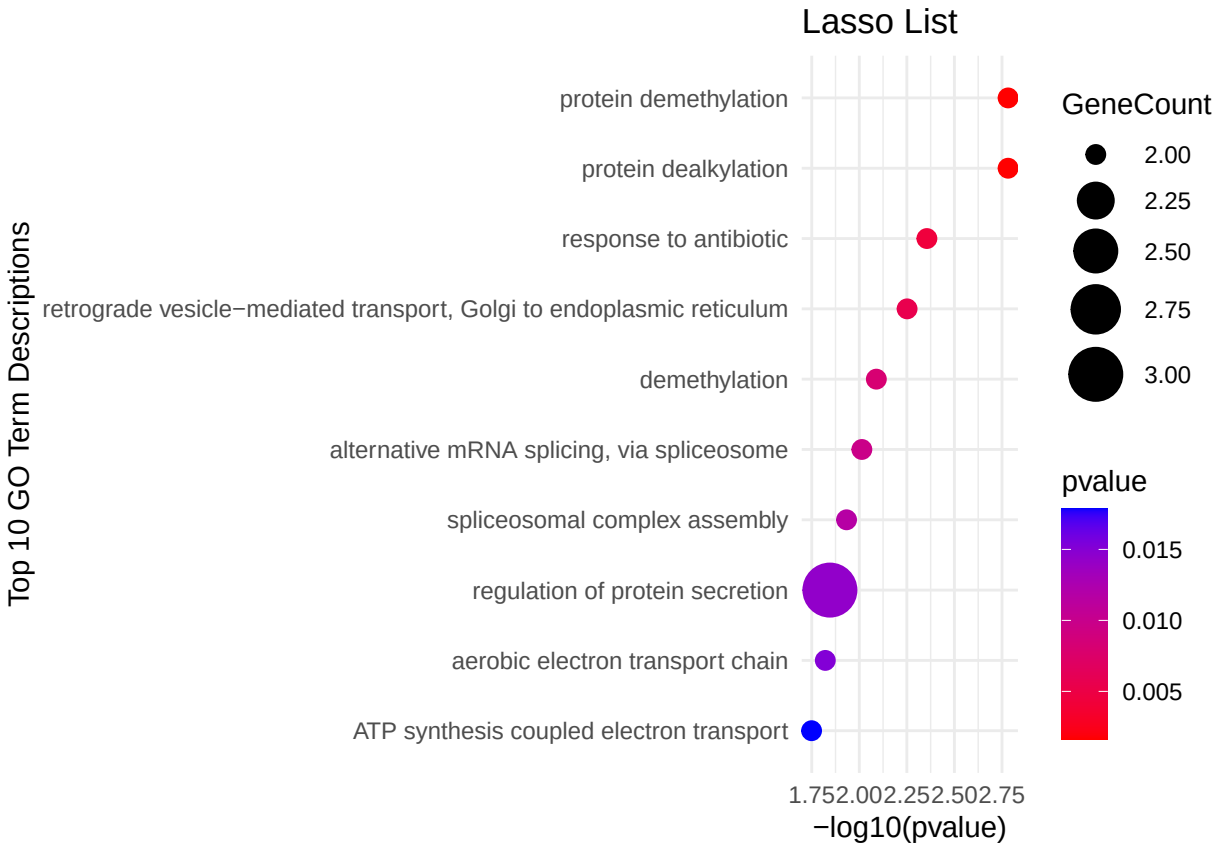
```
# perform GO Analysis
annotated_GO_inbuilt <- GO_enrichment_analysis(annotations_df,
                                      list_type = 'inbuilt', #run GO enrichment on inbuilt  selected f
                                      keyType = 'ENSEMBL', # run analysis with ENSEMBLIDs
                                      background = background,
                                      ont = 'BP') # run BP ontology
#> Performing GO Enrichment analysis for the:Lasso
#> Performing GO Enrichment analysis for the:Univariate
#> Performing GO Enrichment analysis for the:RandomForest
#> Performing GO Enrichment analysis for the:boruta
#> Performing GO Enrichment analysis for the:DEGs

annotated_GO_permutation <- GO_enrichment_analysis(annotations_df,
```

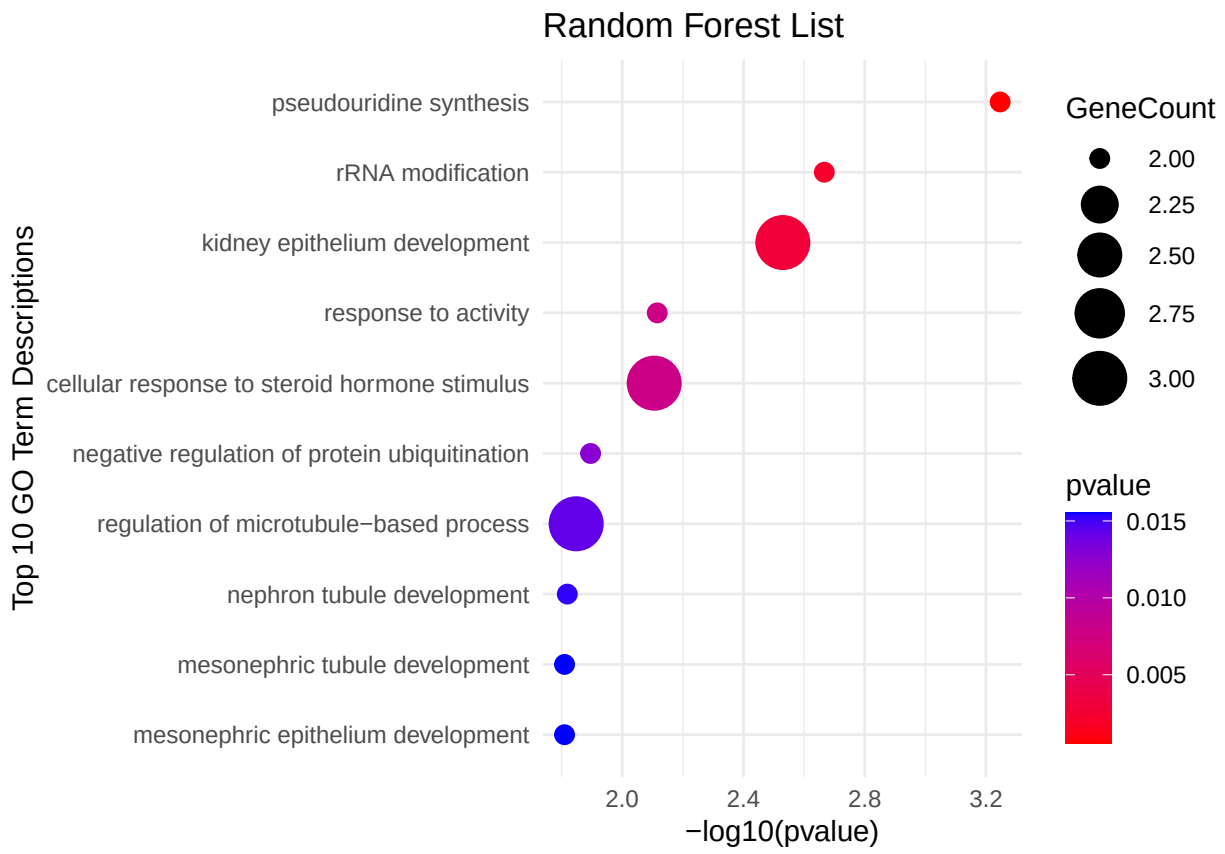
```
                                                    list_type = 'permutation', #run GO enrichment on permuta
                                                    keyType = 'ENSEMBL', # run analysis with ENSEMBLIDs
                                                    background = background,
                                                    ont = 'BP') # run BP ontology
#> Performing GO Enrichment analysis for the:Lasso
#> Performing GO Enrichment analysis for the:Univariate
#> Performing GO Enrichment analysis for the:RandomForest
#> Performing GO Enrichment analysis for the:boruta
#> Performing GO Enrichment analysis for the:DEGs
```

So far inbuilt feature importance looked promising, so now we can inspect what GO terms are enriched in every gene list:

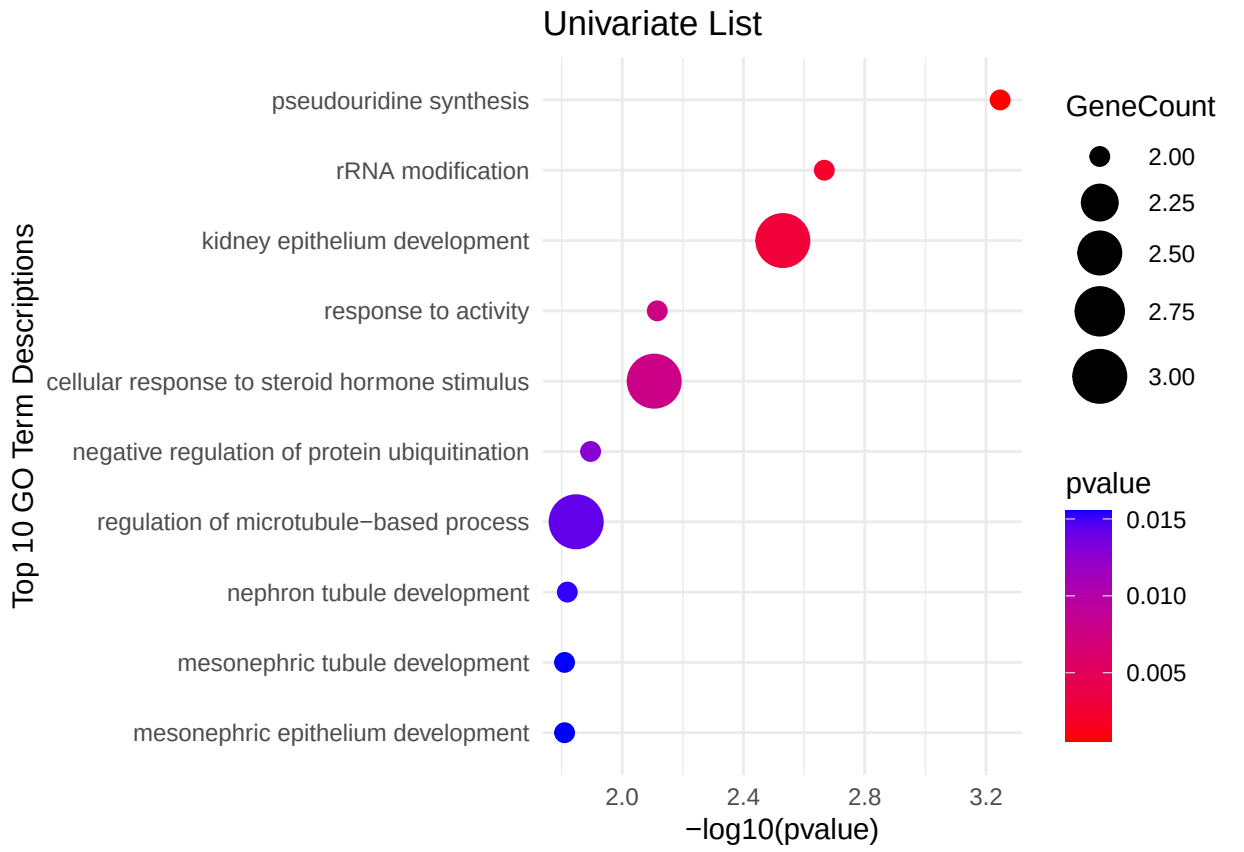



DEGs List
Top 10 GO Term Descriptions
keratinization
epidermis development
regulation of hormone levels
intermediate filament organization
negative regulation of peptidase activity
antimicrobial humoral response
skin development
intermediate filament cytoskeleton organization
intermediate filament−based process
signal release
8
9
10
11
12
−log10(pvalue)
pvalue
3e−08
2e−08
1e−08
GeneCount
20
30
40
50
60
70


Lasso List
Top 10 GO Term Descriptions
protein demethylation
protein dealkylation
response to antibiotic
retrograde vesicle−mediated transport, Golgi to endoplasmic reticulum
demethylation
alternative mRNA splicing, via spliceosome
spliceosomal complex assembly
regulation of protein secretion
aerobic electron transport chain
ATP synthesis coupled electron transport
1.75
2.00
2.25
2.50
2.75
−log10(pvalue)
GeneCount
2.00
2.25
2.50
2.75
3.00
pvalue
0.015
0.010
0.005




```
ggplot(data = annotated_GO_inbuilt$RandomForest@result %>%
         arrange(pvalue) %>%
         head(10) %>%
         mutate(GeneCount = as.numeric(gsub("/.*$", "", GeneRatio))),
       aes(x = reorder(Description, -log10(pvalue)), y = -log10(pvalue))) +
  geom_point(aes(size = GeneCount, color = pvalue)) +
  scale_color_gradient(low = "red", high = "blue") +
  scale_size_continuous(range = c(3, 9)) +
  #geom_text(aes(label = GeneRatio), vjust = -1) +
  coord_flip() +
  xlab("Top 10 GO Term Descriptions") +
  ylab("-log10(pvalue)") +
  ggtitle('Univariate List') +
  theme_minimal()
```

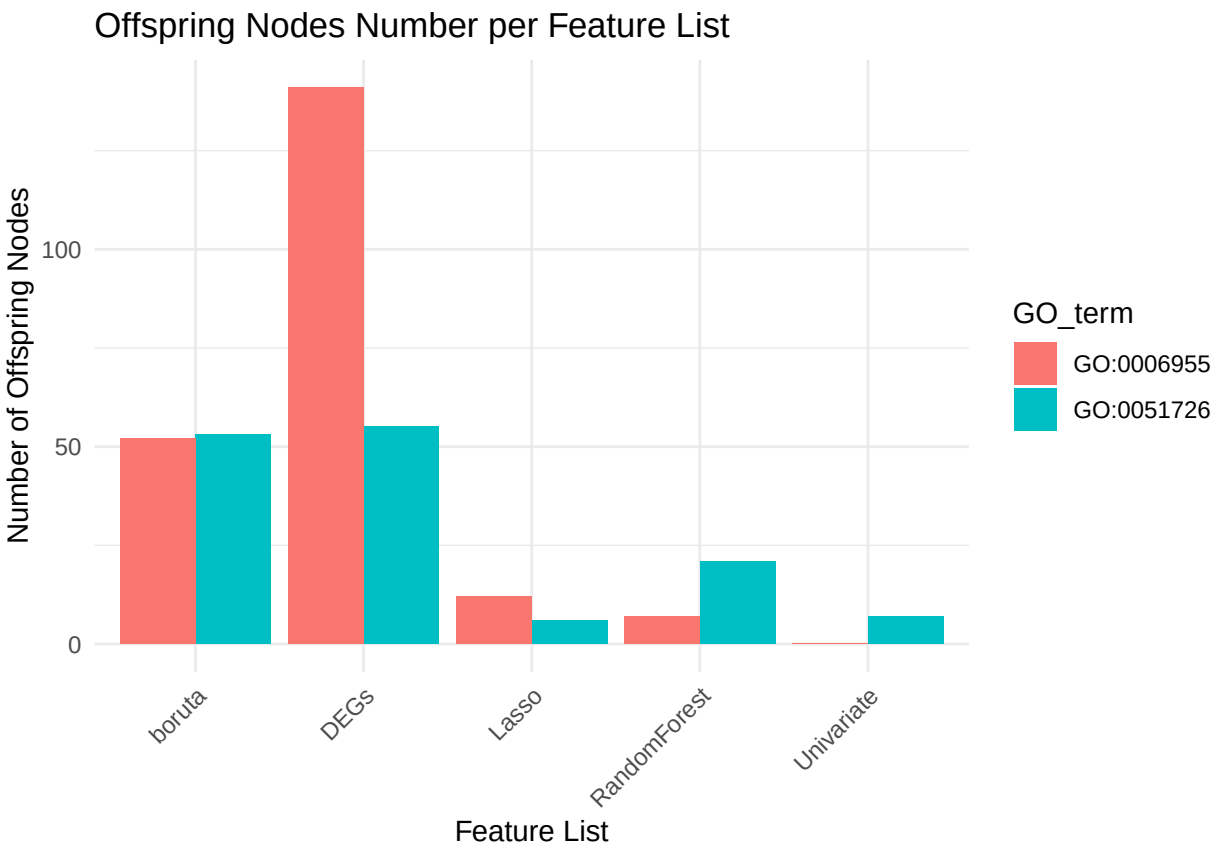


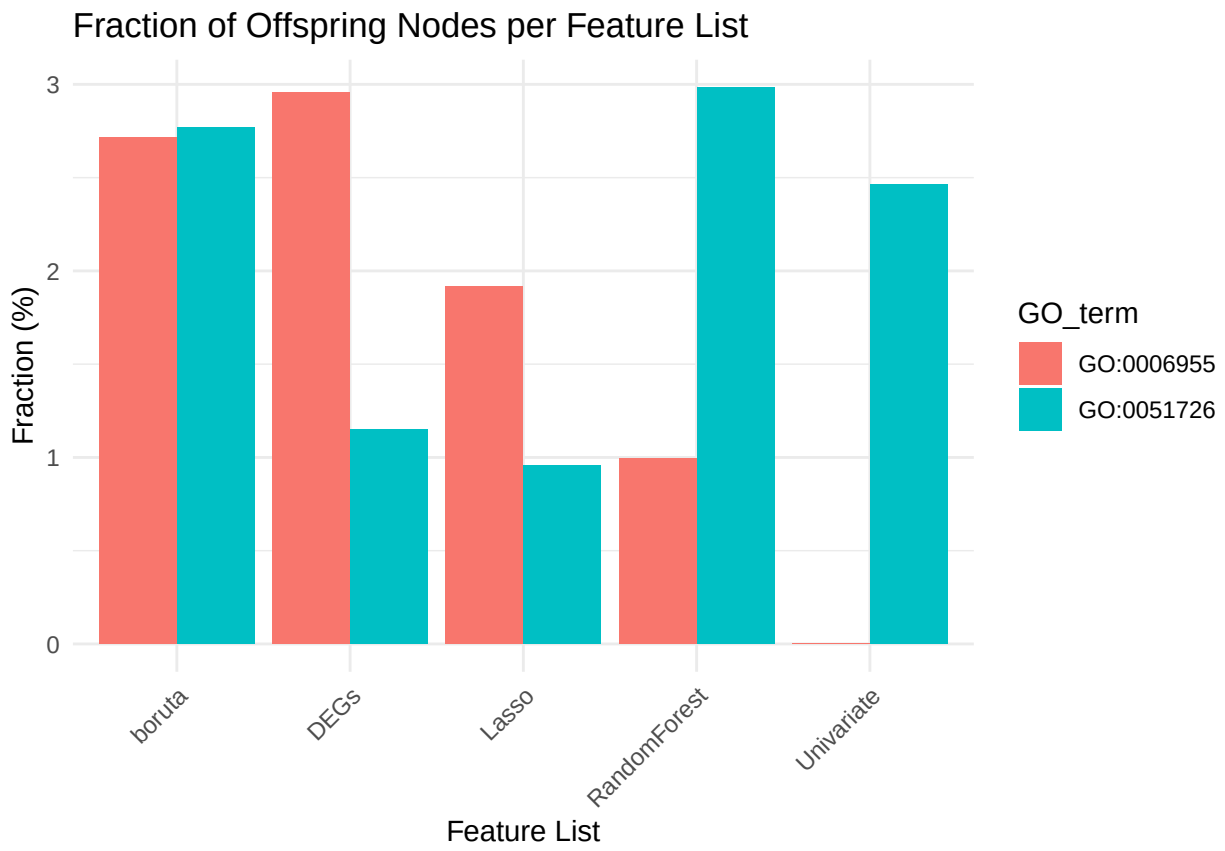
Looking at top 10 enriched pathways, boruta seems to be the one that has the most relevant ones. For example, we can see here sprouting angiogenesis regulation terms that is related to metastasis spread. Additionally, such terms as glycosaminoglycan metabolic process and especially apoptotic p53 pathway are all of high relevance in relation to cancer.

3.6 Quantification of Children Nodes of Parent Node of Interest

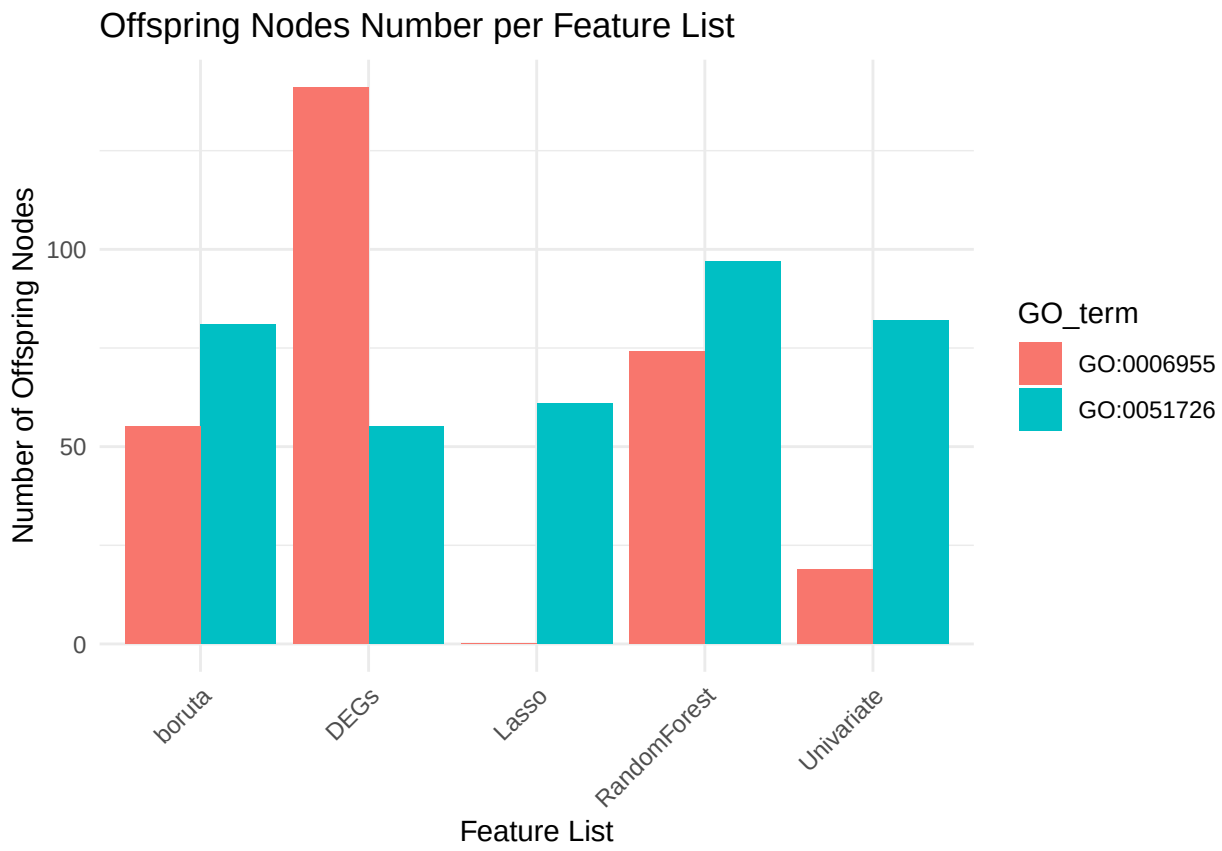
Additionally, we can quantify how many of the children nodes of a parent node of interest are there in our list. For example, we can take two relevant broad GO Biological Process (BP) terms that are cell cycle regulation (GO:0051726) and immune response (GO:0006955).

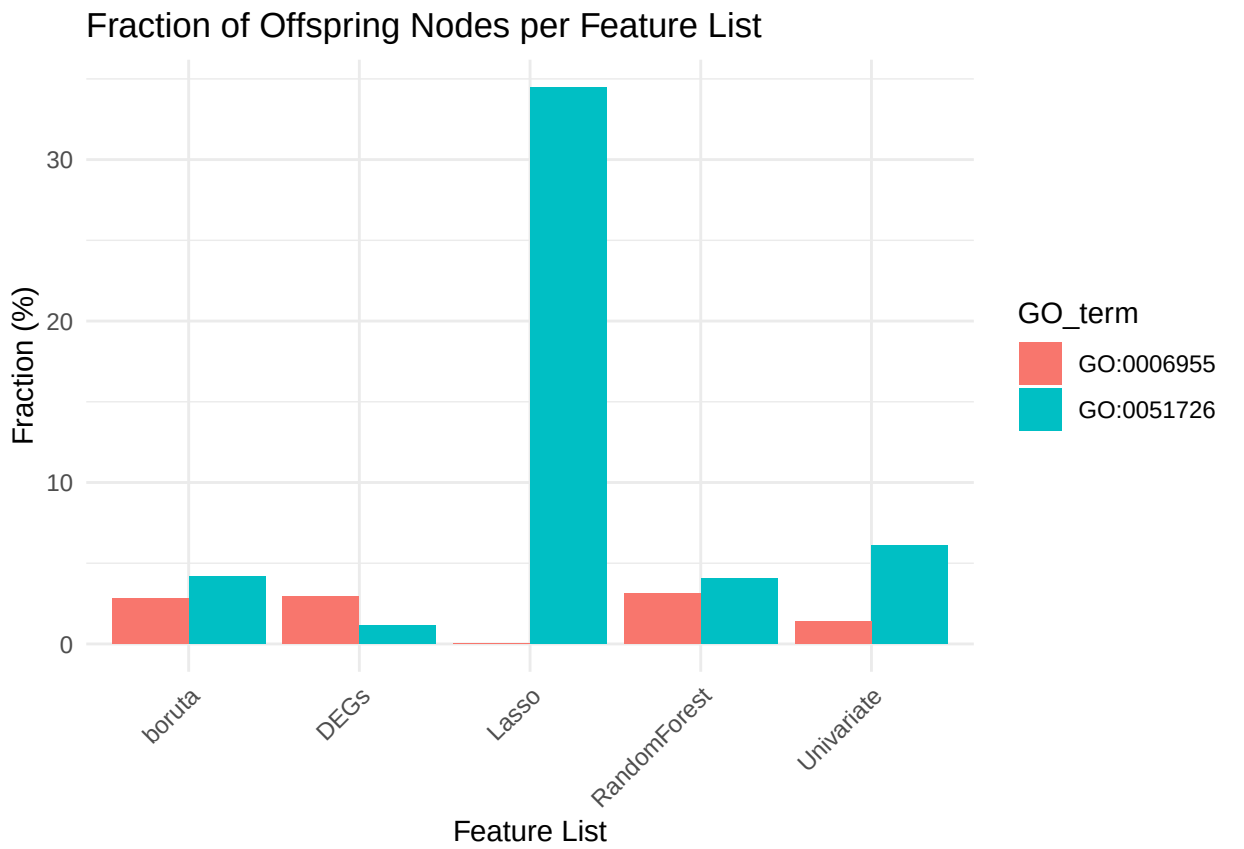
```
annot_child_fractions_inbuilt <- compute_GO_child_term_metrics(GO_data = annotated_GO_inbuilt,
                                 GO_terms = c("GO:0051726", "GO:0006955"),
                                 plot = TRUE)
```


Offspring Nodes Number per Feature List
Number of Offspring Nodes
100
50
0
boruta
DEGs
Lasso
RandomForest
Univariate
Feature List
GO_term
GO:0006955
GO:0051726




```
annot_child_fractions_permut <- compute_GO_child_term_metrics(GO_data = annotated_GO_permutation,
                                GO_terms = c("GO:0051726", "GO:0006955"),
                                plot = TRUE)
```


Offspring Nodes Number per Feature List
Number of Offspring Nodes
100
50
0
boruta
DEGs
Lasso
RandomForest
Univariate
Feature List
GO_term
GO:0006955
GO:0051726

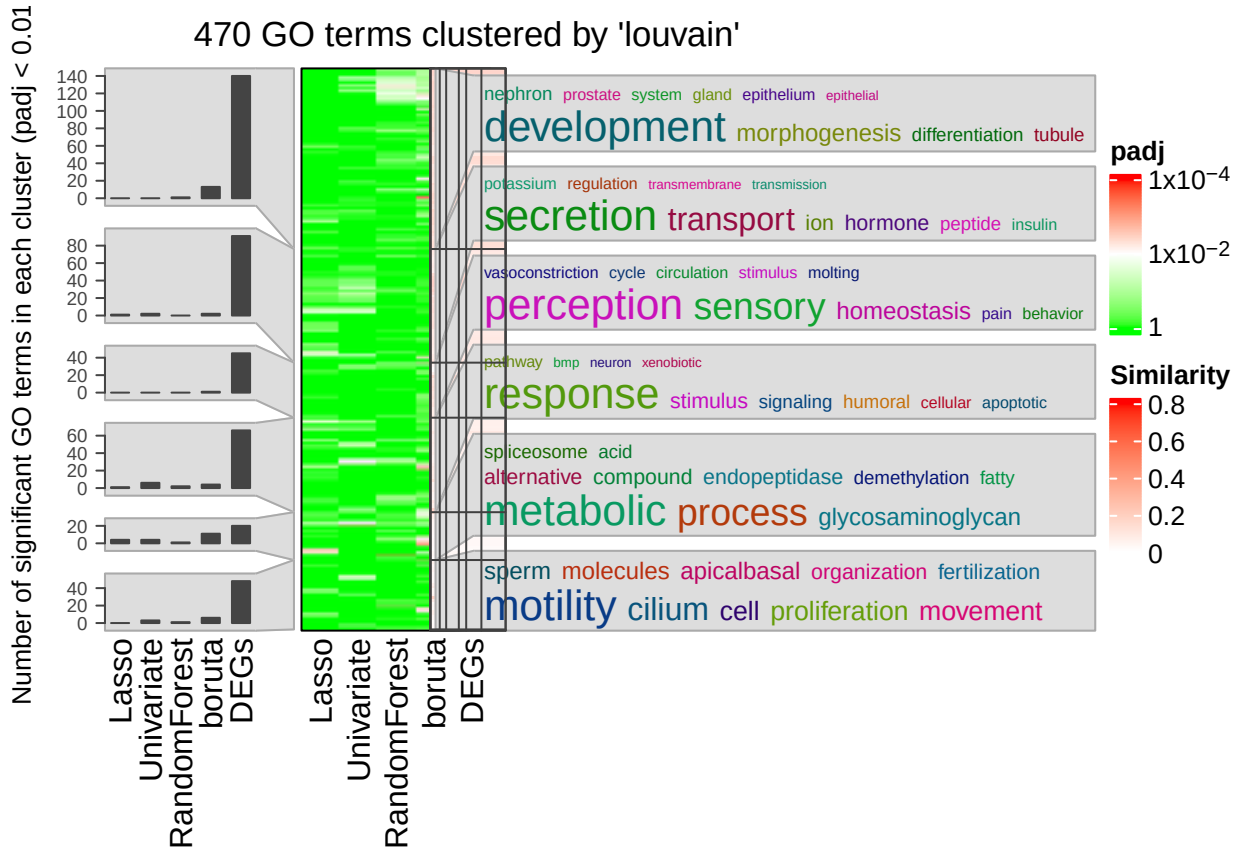


Here, once again we see that in terms of fractions of the parent terms of interest, boruta seems to be performing very well for both 'immune process' and 'cell cycle regulation terms'.

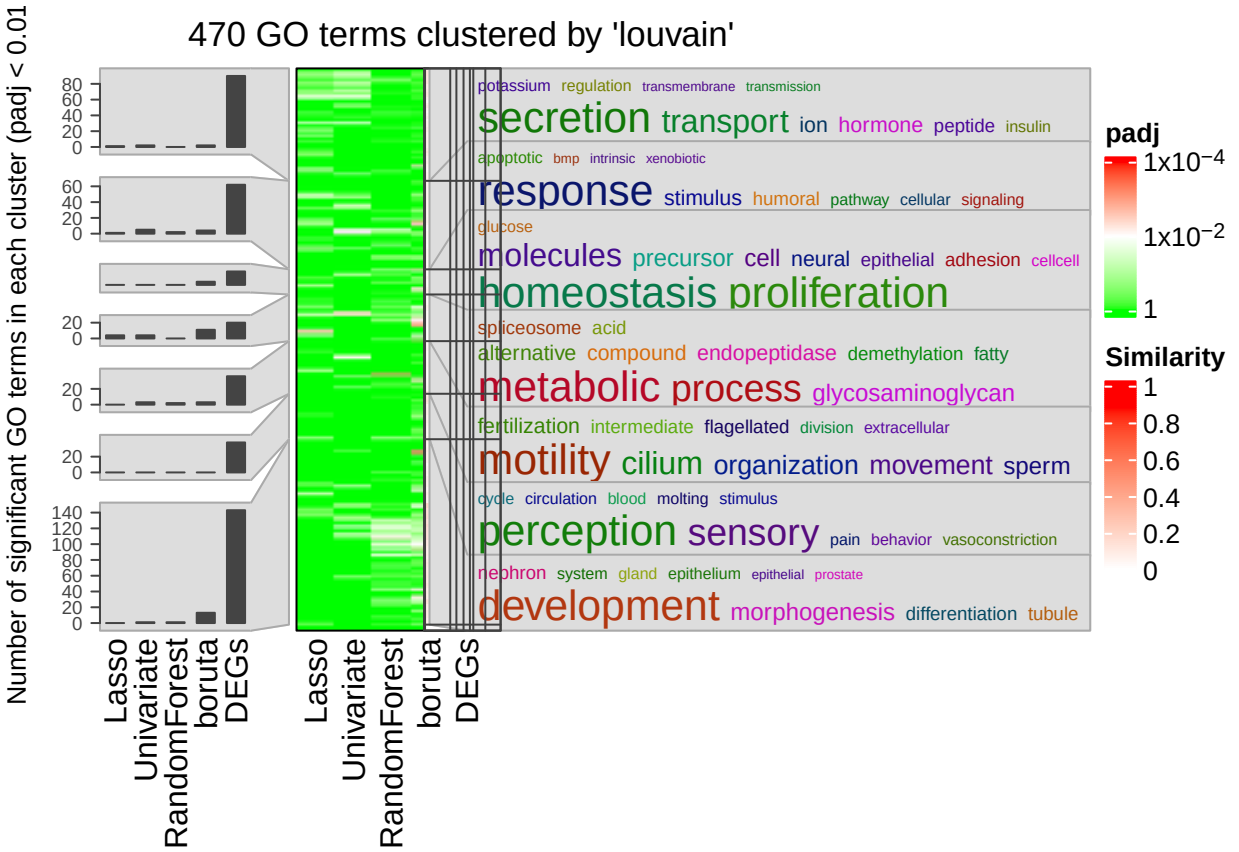
3.7 Semantic Similarity Analysis

Finally, we can now perform semantic similarity (SS) analysis of identified GO terms. To do that we will run this:

```
# perform SS analysis
hmap_inbuilt <- run_simplify_enrichment(annotated_GO_inbuilt,
                                method = 'louvain',
                                measure = 'Rel',
                                ont = 'BP',
                                padj_column = 'pvalue',
                                padj_cutoff = 0.01)
#> Use column 'ID' as `go_id_column`.
#> Loading required namespace: gridtext
#> 470/5074 GO IDs left for clustering.
#> Cluster 470 terms by 'louvain'... 7 clusters, used 0.07406878 secs.
#> Perform keywords enrichment for 6 GO lists...
```




```
hmap_permutation <- run_simplify_enrichment(annotated_GO_permutation,
                                            method = 'louvain',
                                            measure = 'Jiang',
                                            ont = 'BP',
                                            padj_column = 'pvalue',
                                            padj_cutoff = 0.01)
#> Use column 'ID' as `go_id_column`.
#> 470/5074 GO IDs left for clustering.
#> Cluster 470 terms by 'louvain'... 9 clusters, used 0.03672004 secs.
#> Perform keywords enrichment for 7 GO lists...
```



After performing the SS analysis, we can observe that there are 6 clusters in total. By examining word clouds there are two clusters of interest: cluster 4 containing humoral immunity and cell signal transduction terms, and cluster 5 that contains terms related to glycosaminoglycan and other metabolism-related terms. If we look at the signifcance and fraction heatmap on the left we see that while DEGs list is significant everywhere, without any specificity, boruta is singificantly enriched in these two clusters.

4. Picking the Winner and Conclusion

After having completed the entire analysis we can now pick the winner among the lists. From different points of view boruta seems to be the best because: 1. It has highest ML performance on both CV and test sets; 2. Boruta has more relevant GO BP terms for differentiation between molecular subtypes that are related to cancer cell metabolism, apoptosis and angiogenesis; 3. It has higher fractions of parent terms of interest in comparison to other lists;
4. In terms of Semantic Similarity analysis it is more specific to immune response-related terms and metabolism terms, as compared to DEGs that result in broad GO BP terms; 5. In comparison to DEGs list it is much shorter and more manageable for the downstream analysis and interpretation With this boruta seems to be the best candidate for downstream analysis and probably targeted experiments to establish biomarkers. With all this in mind, we can clearly say that the most suitable list for the further investigation is boruta.

```
utils::sessionInfo()
#> R version 4.3.1 (2023-06-16 ucrt)
#> Platform: x86_64-w64-mingw32/x64 (64-bit)
#> Running under: Windows 11 x64 (build 22000)
#>
```

```
#> Matrix products: default
#>
#>
#> locale:
#> [1] LC_COLLATE=Russian_Kazakhstan.utf8  LC_CTYPE=Russian_Kazakhstan.utf8
#> [3] LC_MONETARY=Russian_Kazakhstan.utf8 LC_NUMERIC=C
#> [5] LC_TIME=Russian_Kazakhstan.utf8
#>
#> time zone: Europe/Paris
#> tzcode source: internal
#>
#> attached base packages:
#> [1] stats4    stats     graphics  grDevices utils     datasets  methods
#> [8] base
#>
#> other attached packages:
#>  [1] ensembldb_2.24.0            AnnotationFilter_1.24.0
#>  [3] GenomicFeatures_1.52.1      AnnotationDbi_1.62.2
#>  [5] ggplot2_3.4.2               DESeq2_1.40.2
#>  [7] SummarizedExperiment_1.30.2 Biobase_2.60.0
#>  [9] MatrixGenerics_1.12.2       matrixStats_1.0.0
#> [11] GenomicRanges_1.52.0        GenomeInfoDb_1.36.1
#> [13] IRanges_2.34.1              S4Vectors_0.38.1
#> [15] BiocGenerics_0.46.0         GeneSelectR_0.0.0.9000
#> [17] dplyr_1.1.2
#>
#> loaded via a namespace (and not attached):
#>   [1] ProtGenerics_1.32.0           bitops_1.0-7
#>   [3] enrichplot_1.20.0             HDO.db_0.99.1
#>   [5] httr_1.4.6                    RColorBrewer_1.1-3
#>   [7] doParallel_1.0.17             tools_4.3.1
#>   [9] utf8_1.2.3                    R6_2.5.1
#>  [11] lazyeval_0.2.2                GetoptLong_1.0.5
#>  [13] withr_2.5.0                   prettyunits_1.1.1
#>  [15] gridExtra_2.3                 plotwidgets_0.5.1
#>  [17] cli_3.6.1                     scatterpie_0.2.1
#>  [19] labeling_0.4.2                slam_0.1-50
#>  [21] tm_0.7-11                     commonmark_1.9.0
#>  [23] Rsamtools_2.16.0              yulab.utils_0.0.6
#>  [25] gson_0.1.0                    DOSE_3.26.1
#>  [27] rstudioapi_0.15.0             RSQLite_2.3.1
#>  [29] generics_0.1.3                gridGraphics_0.5-1
#>  [31] shape_1.4.6                   BiocIO_1.10.0
#>  [33] gtools_3.9.4                  GO.db_3.17.0
#>  [35] Matrix_1.5-4.1                fansi_1.0.4
#>  [37] lifecycle_1.0.3               yaml_2.3.7
#>  [39] gplots_3.1.3                  qvalue_2.32.0
#>  [41] BiocFileCache_2.8.0           grid_4.3.1
#>  [43] blob_1.2.4                    promises_1.2.0.1
#>  [45] crayon_1.5.2                  lattice_0.21-8
#>  [47] cowplot_1.1.1                 KEGGREST_1.40.0
#>  [49] magick_2.7.5                  pillar_1.9.0
#>  [51] knitr_1.43                    ComplexHeatmap_2.16.0
```

```
#>  [53] fgsea_1.26.0                   rjson_0.2.21
#>  [55] codetools_0.2-19               fastmatch_1.1-3
#>  [57] glue_1.6.2                     downloader_0.4
#>  [59] ggfun_0.1.1                    data.table_1.14.8
#>  [61] vctrs_0.6.3                    png_0.1-8
#>  [63] treeio_1.24.1                  testthat_3.1.10
#>  [65] gtable_0.3.3                   cachem_1.0.8
#>  [67] xfun_0.39                      S4Arrays_1.0.4
#>  [69] mime_0.12                      tidygraph_1.2.3
#>  [71] pheatmap_1.0.12                iterators_1.0.14
#>  [73] interactiveDisplayBase_1.38.0 ellipsis_0.3.2
#>  [75] nlme_3.1-162                   simplifyEnrichment_1.10.0
#>  [77] ggtree_3.8.0                   bit64_4.0.5
#>  [79] progress_1.2.2                 filelock_1.0.2
#>  [81] KernSmooth_2.23-22             colorspace_2.1-0
#>  [83] DBI_1.1.3                      tidyselect_1.2.0
#>  [85] proxyC_0.3.3                   bit_4.0.5
#>  [87] compiler_4.3.1                 curl_5.0.1
#>  [89] xml2_1.3.5                     NLP_0.2-1
#>  [91] DelayedArray_0.26.6            shadowtext_0.1.2
#>  [93] rtracklayer_1.60.0             scales_1.2.1
#>  [95] caTools_1.18.2                 rappdirs_0.3.3
#>  [97] stringr_1.5.0                  digest_0.6.33
#>  [99] tmod_0.50.13                   rmarkdown_2.23
#> [101] XVector_0.40.0                 htmltools_0.5.5
#> [103] pkgconfig_2.0.3                highr_0.10
#> [105] dbplyr_2.3.3                   fastmap_1.1.1
#> [107] rlang_1.1.1                    GlobalOptions_0.1.2
#> [109] shiny_1.7.4.1                  farver_2.1.1
#> [111] jsonlite_1.8.7                 BiocParallel_1.34.2
#> [113] GOSemSim_2.26.1                RCurl_1.98-1.12
#> [115] magrittr_2.0.3                 GenomeInfoDbData_1.2.10
#> [117] ggplotify_0.1.1                patchwork_1.1.2
#> [119] munsell_0.5.0                  Rcpp_1.0.11
#> [121] ape_5.7-1                      viridis_0.6.3
#> [123] reticulate_1.32.0.9000         stringi_1.7.12
#> [125] tagcloud_0.6                   ggraph_2.1.0
#> [127] brio_1.1.3                     zlibbioc_1.46.0
#> [129] MASS_7.3-60                    AnnotationHub_3.8.0
#> [131] plyr_1.8.8                     org.Hs.eg.db_3.17.0
#> [133] parallel_4.3.1                 ggrepel_0.9.3
#> [135] Biostrings_2.68.1              graphlayouts_1.0.0
#> [137] splines_4.3.1                  gridtext_0.1.5
#> [139] hms_1.1.3                      circlize_0.4.15
#> [141] locfit_1.5-9.8                 igraph_1.5.0
#> [143] markdown_1.7                   reshape2_1.4.4
#> [145] biomaRt_2.56.1                 BiocVersion_3.17.1
#> [147] XML_3.99-0.14                  evaluate_0.21
#> [149] RcppParallel_5.1.7             BiocManager_1.30.21
#> [151] foreach_1.5.2                  tweenr_2.0.2
#> [153] httpuv_1.6.11                  tidyr_1.3.0
#> [155] purrr_1.0.1                    polyclip_1.10-4
#> [157] clue_0.3-64                    ggforce_0.4.1
```

```
#> [159] xtable_1.8-4                  restfulr_0.0.15
#> [161] tidytree_0.4.2                later_1.3.1
#> [163] viridisLite_0.4.2             tibble_3.2.1
#> [165] clusterProfiler_4.8.2         aplot_0.1.10
#> [167] memoise_2.0.1                 beeswarm_0.4.0
#> [169] GenomicAlignments_1.36.0      cluster_2.1.4
```